Performance Evaluation of Reconfiguration Algorithms for the Reconfigurable Network on Chip Architecture RecMIN

Alexander Logvinenko, Dietmar Tutsch
University of Wuppertal
Emails: alexanderlogv@gmail.com, tutsch@uni-wuppertal.de

Abstract—**The Reconfigurable Multi-Interconnection Network (RecMIN) is a new network architecture that reduces inefficiency and increases the throughput of the network on chip. The RecMIN topology adapts itself to traffic flow by reconfiguration. Three reconfiguration algorithms are employed, in order to take advantage of the capabilities of the RecMIN architecture. The η-algorithm, the minimal queues algorithm and the pattern identification algorithm allow the network to adapt itself to different traffic distributions. Furthermore, an observation technique that notes changes in traffic pattern is presented, in order to avoid infinite reconfiguration processes. The performance of the algorithms is presented.**

Keywords-Network on Chip; Reconfiguration Algorithms; Reconfiguration Architecture.

I. Introduction

Modern Systems on Chip (SoC) are built so that they consist of many independent individually designed units (Intellectual Property cores or IP- cores), e.g., cache memory, I/O controllers, audio/video interfaces, etc. Buses were used up to now in order to enable communication among these units. Today, however, designers prefer Networks on Chip (NoC) for efficient interaction among IP-cores. Therefore, the speed and efficiency of modern SoC depend not just on the speed of single units of IP but also on the properties of the NoC used [1]. The main properties of the NoC are source output, target throughput and packet delay. The latter ones depend not just on topology of the network, routing algorithm, buffering strategy, packet switching but also on how efficiently network operates in case of bottlenecks during the packet traffic flow.

The popular solution to solve the problem of a partially overloaded network (bottlenecks) due to inefficiency, is to implement a complex algorithm that reroutes data flow in NoC. The complexity of such algorithms usually grows exponentially with the size of network. So, as an alternative to the re-routing algorithms, some works from the academic community have been focusing on the possibility of adopting NoC by reconfiguration.

For instance, Tutsch and Lüdtke [2][3][4] and Al Faruque [5][6] suggest that the directions of data flow should be changed in order to optimize the NoC for special traffic profiles. Unlike them, this paper continues the previously [7][8][9] introduced topic of the RecMIN architecture. In this article, however, we present three different algorithms for the optimization of a network that uses the RecMIN architecture.

The paper is structured as following: Section 2 introduces the reconfiguration architecture RecMIN. Section 3 deals with the η-function, which enables to evaluate of the network-on-chip performance. In most important Section 4, three algorithms are presented and compared: η algorithm, minimal ques algorithm, pattern identification algorithm. Section 5 concludes.

II. Reconfiguration Architecture RecMIN

The main problem of NoC as compared to the full connection of all inputs/outputs is the chance of bottlenecks to arise given certain traffic structures. In this work, Multistage Interconnection Metwork (MIN) architecture is used, which is built out of 2×2 routers [10]. An example of this kind of network is shown in Fig. 1. The technical realisation of MIN topology is given, e.g., in [11] and [12]. One of the characteristics of MIN is that all the traffic loads have to pass through all the stages of the MIN. Especially for asymmetrical traffic, the connection wires between the stages can lead to tailbacks.

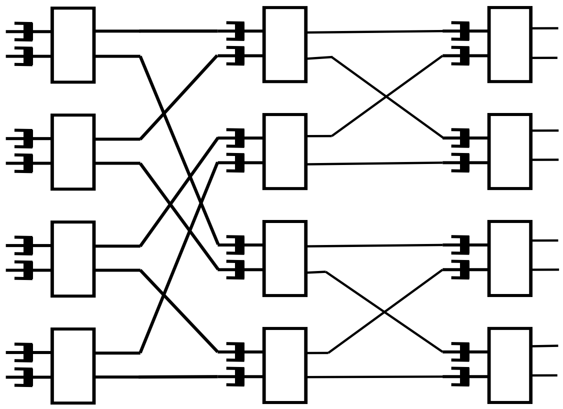

Figure 1. MIN architecture with 8x8 inputs/outputs built out of 2x2 routers

Reconfiguration architecture RecMIN solves the problem of bottlenecks in two out of three possible cases. The proposal is to create the MIN not from the 2x2 routers, as usual, but from specific reconfiguration half cells - Reconfiguration Half Cell (RecHC). The architecture of this cell is given in Fig. 2.

RecHC has 8 inputs and 8 outputs. In front of each input, one buffer element is located. Each half-cell can be used in one of these two possible modes: In the first mode (Mode A), the RecHC consists of four independent 2x2 routers. In the

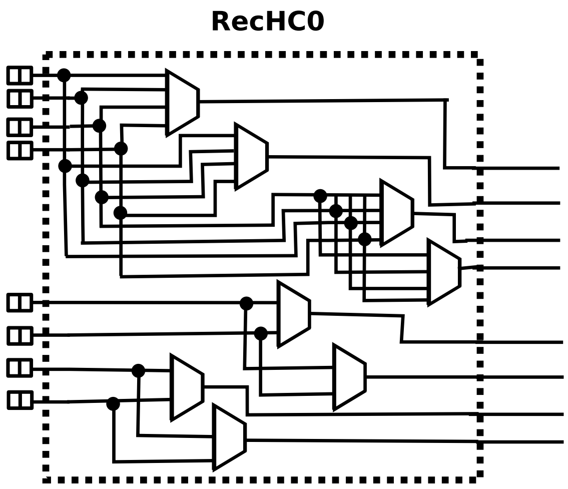

Figure 2. The architecture of reconfiguration half cell - RecHC

second mode (Mode B), there is however just one 4x4 router in the upper part of the cell, and four simple wire connections without any logic in its bottom part. If a RecHC changes the mode from A to B (Fig. 3), then packets which arrive in the upper part of the Half Cell are distributed correctly without problems. Though, in the bottom part of the RecHC problems may arise, since in the mode B no redirection takes place, and the packets are transferred straight forward (Fig. 3). For example, after switching from mode A to mode B some packets in buffer of input i4 that are addressed to output o5 have no possibility to arrive at their targets (e.g., IP-cores). Therefore, usage of two half cells simultaneously is to be preferred.

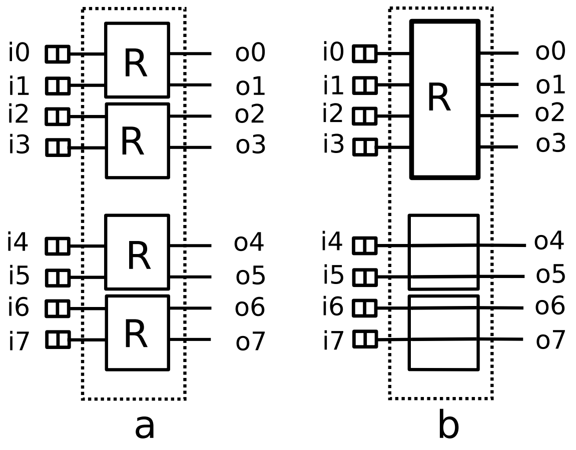

Figure 3. RecHC in two modes. a: mode A, b: mode B

The two RecHCs are put together (the second one upside down), to form one reconfiguration cell - RecCell (Fig. 4). If both of RecHCs that build RecCell are put into the Mode A, then the construction leads to two independent MINs (Fig. 4a), with 4x4 inputs-outputs and 2x2 routers each. If the two RecHCs are put in mode B, then two independent 4x4 routers emerge (Fig. 4b). The other two combinations (AB and BA) are meaningless and therefore are not used. So, a full cell has two possible reconfigurations: folded (BB) and unfolded (AA).

With RecCells, it is possible to build a MIN. The resulting structure is called RecMIN. If the number of 2x2 switches

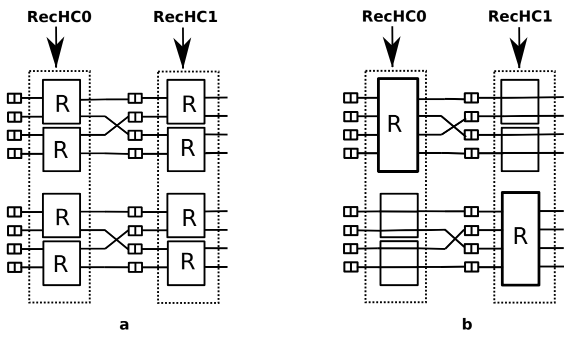

Figure 4. RecCell in two modes. a: unfolded mode, b: folded mode

in MIN is divisible by 16, the entire network can be built from reconfiguration cells. Otherwise, it is necessary to use two non-reconfigurable 2x2 switches in order to connect the reconfiguration cells. Therefore, RecMIN with an arbitary number of 2x2 routers can be implemented.

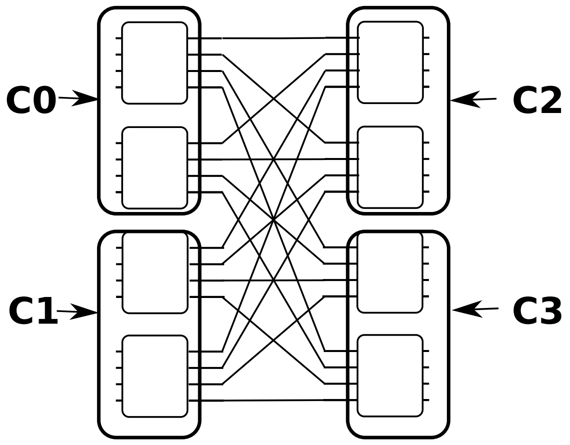

Figure 5. RecMIN with 16 inputs/outputs

In this paper, the RecMIN with 16 inputs outputs is used as an example for RecMIN architecture (Fig. 5). This RecMIN can be build out of four RecCells: C0, C1, C2 and C3.

If we compare our architecture (Fig. 6) with the one of the non-reconfigurable MIN with 2x2 routers, we will see that the dotted line marked router connection (between the first and the second stage of the 2x2 routers) can be reconfigured. So, if the traffic in the network generates bottlenecks in these places, the NoC can reconfigure its topology according to the adaptation of the architecture to the traffic load, and so increase the throughput of the network and decrease the packet delay.

It can be said that if the traffic unfortunately generates a bottleneck in one of the non-reconfigurable wires, the reconfiguration will not help. But, usually, the designer of the NoC knows the application for which the network is to be designed, and so can pre-arrange the most expected bottleneck-wires inside the reconfiguration cells.

The other way around, the 4x4 router would have less throughput than a 2x2 router [13]. So, for the symmetrical high load (more than 0.63 flits per clock cycle), the 4x4 routers will automatically become NoC bottlenecks. In this case, a back

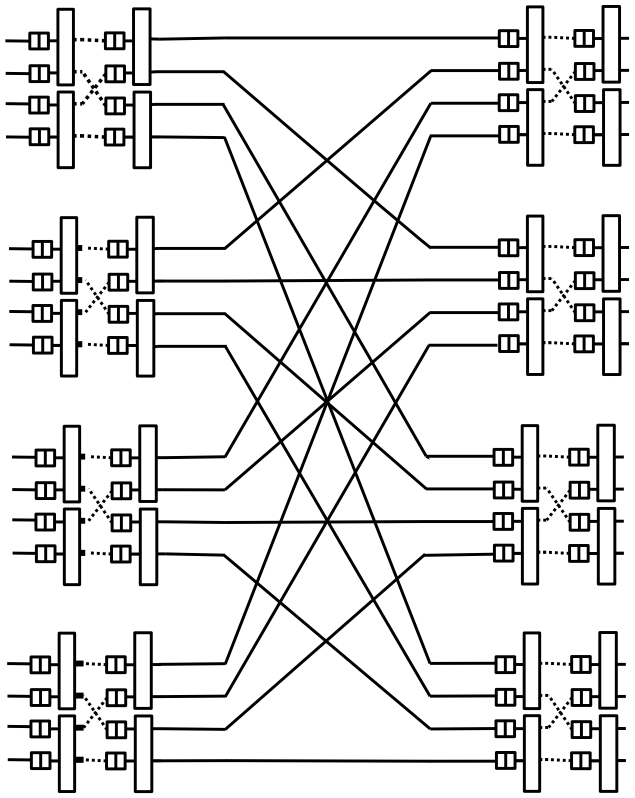
Figure 6. RecMIN with 16 inputs/outputs

reconfiguration of the RecCells to unfolded mode is necessary (see [13]).

The other disadvantage of the architecture is that two independent routers of the NoC are now bonded together. If one of the 4x4 routers of RecCell "decides" to change mode (from unfolded to folded or other way around), it has to check if the other router of the same RecCell will "agree" to change the mode as well.

III. Evaluation of Network Performance

The network performance of asymmetrical NoC for asymmetrical load is measured by three main parameters: throughput of network sources (ς_i, where i is a number of the source in NoC), throughput of network targets (τ_i, where i is a number of the target in NoC), and packet delay for each target (δ_i, where i is a number of the target in NoC). To evaluate network efficiency dependent on the packet load, we use η-function:

$$\eta = \sum_{i=0}^{N-1} (\varsigma_i * C_{\varsigma_i} + \tau_i * C_{\tau_i} + \delta_i * C_{\delta_i}) \qquad (1)$$

where N is the number of sources/targets in NoC and constants C_{ς_i}, C_{τ_i}, C_{δ_i} are priority weights for throughput and delay defined by SoC designer. By setting the priorities the designer specifies how important the corresponding NoC parameter is.

For example, for specific NoC the throughput for sources and targets may be not as important as a minimal delay. Furthermore, packet delay is especially important for the targets T_4 and T_5. In this case, the constants C_{ς_i}, C_{τ_i}, C_{δ_i} can be adjusted as follows:

$$\begin{array}{rll} C_{\varsigma_i} = C_{\tau_i} = 0 & [\text{clock cycles/flit}] & \text{for } i \in \{0,..,N-1\} \\ C_{\delta_i} = -1 & [\text{clock cycles}]^{-1} & \text{for } i \in \{0,..,N-1\}n\{4,5\} \\ C_{\delta_4} = C_{\delta_5} = -2 & [\text{clock cycles}]^{-1} & . \end{array} \qquad (2)$$

The parameter ς_i, τ_i, δ_i depend on the topology of the network, and are calculated using simulation. By simulating the different network reconfigurations, designer is able to compare the performance of different network topologies for specified traffic. Table 1 gives an example for a network with 16 inputs and 16 outputs consisting of four RecCells (Fig. 6). Constants of priorities are chosen according to (2).

In Table 1, G$_i$ is the notation for each source (generator) i; P$_{tr}$ (G$_i$) is the probability that the source i sends a packet per clock time unit; P$_{rec}$ (T$_x$) is the probability that the target node x receives a packet from the generator i (G$_i$) per clock time unit.

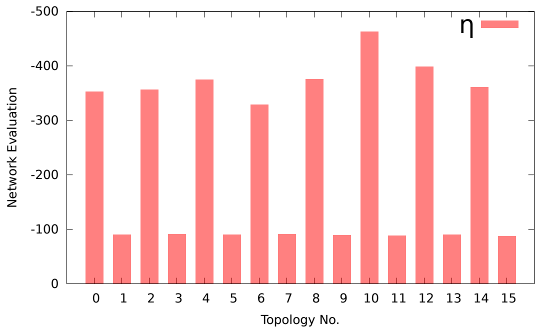

Figure 7. η-function for 16x16 RecMIN loaded with traffic from Table 1

The simulation results are presented in Fig. 7 (Simulation parameters for all simulations presented in this paper are: buffer size 16 phits for each buffer, conflict resolution algorithm for each router is "random choice", each packet consists of one flit, a flit equals the size of a phit). It shows the evaluation of different RecMIN reconfigurations, resulting from all possible RecCell modes. The used NoC consists of 4 RecCells each of them can be used in two possible modes, so, for this kind of network there exist $2^4 = 16$ possible topologies. As is shown in Fig. 7, η-function has the highest rates for topologies with an odd number (1,3,5 etc.), and the lowest rate for topology 10. Thus, for the optimal communication of NoC components by traffic defined in Table 1, RecMIN must be reconfigured to topologies 1,3,5,7,9,11,13, or 15.

IV. Reconfiguration Algorithms

It is not sufficient only to offer a reconfigurable architecture when considering the reconfiguration of NoC as an opportunity to improve its efficiency and performance. A second step is required in order to take advantage of the capabilities of the architecture: employing algorithms that allow the network to

TABLE I. LOAD IN RECMIN

Generator	$\mathbf{P}_{tr}(\mathbf{G}_i)$	$\mathbf{P}_{rec}(\mathbf{T}_{10})$	$\mathbf{P}_{rec}(\mathbf{T}_{11})$	$\mathbf{P}_{rec}(\mathbf{T}_{12})$	$\mathbf{P}_{rec}(\mathbf{T}_{rst})$
G_0, G_1	0.4875	0.2/16	0.3	0.2/16	0.2/16
G_2, G_3	0,3875	0.2/16	0.2/16	0.2	0.2/16
G_4 - G_7	0,55	0.1	0.2/16	0.2/16	0.2/16
G_8 - G_{15}	0.2	0.2/16	0.2/16	0.2/16	0.2/16

adapt itself to different traffic distributions. In this paper, we propose several algorithms that were developed for RecMIN architecture: the η-algorithm, the minimal queues algorithm and the pattern identification algorithm.

A. General requirements for algorithms

The tasks of the algorithm responsible for the reconfiguration of the network can be divided into the following steps:

- Monitoring the trigger: tracking events or sequences of events, after which the algorithm has to decide about the reconfiguration of the network topology.
- Looking for bottlenecks: finding parts of the network that need to be changed due to reconfiguration
- Looking for alternative structure: finding a topology to substitute the previous one
- Processing the reconfiguration

Each of these steps should avoid high time consumption and should require simple calculation wherever possible. (Implementation of complex calculations in hardware, requires expensive chip area). Another key issue is the question of stability. It is necessary to avoid a situation where the algorithm constantly tries to optimize the network. Doing so the algorithm continually conducts endless reconfiguration processes, hence preventing the network from operating in normal mode. For example, such a problem can occur if reconfiguration algorithm is unable to find an unambiguously best network topology. Thus, after checking different network reconfigurations the found topology is still not optimal. This state directs to the retriggering of the algorithm thereby starting a new reconfiguration process. Therefore, no reconfiguration process should be started, if the algorithm is unable to find a better NoC topology for the traffic flow unless the network traffic changes. Consequently, it is essential not only to have a trigger to reconfigure the network, but also to implement an observation technique that notes changes in traffic pattern.

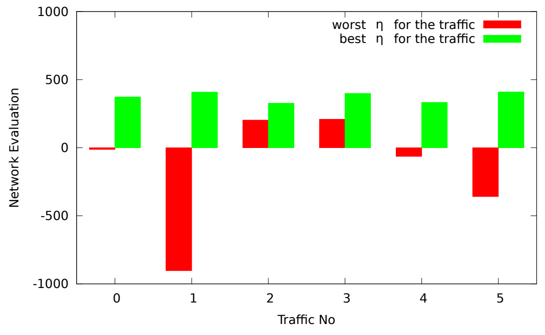

Figure 8. η-functions for different traffics in 16x16 RecMIN

This paper proposes to monitor the traffic flow by changes of queue lengths in the NoC buffers, in order to solve the problem of instability. Assume, each traffic corresponds to a vector $\vec{\theta}$. Thus, change in the traffic flow is observed by $\vec{\Delta\theta}$, the difference between two previous calculated $\vec{\theta}$-vectors:

$$\vec{\Delta\theta} = \vec{\theta_2} - \vec{\theta_1} = \begin{pmatrix} \theta_{0,2} \\ \theta_{1,2} \\ \vdots \\ \theta_{N-1,2} \end{pmatrix} - \begin{pmatrix} \theta_{0,1} \\ \theta_{1,1} \\ \vdots \\ \theta_{N-1,1} \end{pmatrix} \tag{3}$$

where $\theta_{i,j}$ is the length of the queue in the buffer j caused by traffic number i. In the more general case the system records the combination of topologies and corresponding traffic vectors in memory registers. Then, the reconfiguration algorithm can immediately change the NoC to the optimal topology, if the network traffic pattern repeats after some time.

B. The η-Algorithm

The η-algorithm uses the η-function for the evaluation and improvement of the network effectiveness. The algorithm receives the mean values for the network settings (throughput and delay) every 1000 cycles (number of cycles can be changed by the network designer). It calculates the value of η based on these means. If η-value falls below the specified threshold, the algorithm starts the reconfiguration.

Looking for an alternative structure is a typical global optimization problem of locating a good approximation to the global optimum of a given function. We used an exhaustive search of all possible topology reconfigurations, to find the optimal one. It is a reasonable alternative for small networks. (We used 16×16 RecMIN, where only $2^4 - 1$=15 reconfigurations are possible (the original configuration is not a reconfiguration). For networks with the higher number of RecCells, we recommend the usage of simulated annealing, genetic algorithms or other heuristic algorithms). Once all possible topologies for RecMIN have been iterated, the algorithm chooses the one with the maximum η-value.

The η-algorithm written in pseudo-code is shown below:

```
INPUT:  RecMIN, traffic;
OUTPUT: RecMIN_topology;

best_calculated_$\eta$:= calculate $\eta$;
BEGIN
 IF $\eta$<$\eta$_threshold THEN
    IF no reconfiguration is running THEN
     FOR i:=0
       TO i<all_possible_reconfigurations - 1
       DO
            simulate topologyi;
            calculate $\eta$;
            IF calculated $\eta$>best_calculated_$\eta$
            THEN
             best_calculated_$\eta$:=calculated_$\eta$;
```

```
              best_topology:=sim_topology;
            END IF;
       END FOR;
     END IF;
 END IF;
 RETURN best_topology;
END;
```

Advantages and Disadvantages of the η-Algorithm: The main advantage of the η-algorithm is the possibility of finding the optimal network topology for any traffic. Fig. 8 shows the analysis of six NoC traffics (in 16×16 RecMIN consisting of four RecCells) using the η-algorithm. The chosen priority weights are $C_{\varsigma_i} = C_{\tau_i} = 50$ [clock cycles/flit] and $C_{\delta_i} = -1$ [clock cycles]$^{-1}$ for all $i \in \{0, .., N-1\}$. For each traffic, Fig. 8 shows two values: the minimum and maximum value of η, which can be achieved by reconfiguring the network with the η-algorithm.

Fig. 8 shows the result of six chosen traffics given by the η-algorithm. It can be seen that for some traffic flows (e.g., traffic nos. 1 and 5) it is possible to achieve a good improvement of network performance by reconfiguration. On the other hand, some traffics exist (e.g., traffic nos. 2 and 3), for which reconfiguration does not lead to distinct enhancements. Therefor, the usage of the η-algorithm with exhaustive search is not reasonable for this kinds of traffics.

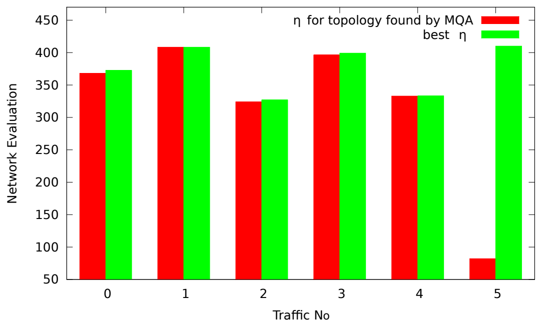

Figure 9. Comparison between the MQA and the η-algorithm for different traffics in 16x16 RecMIN

The additional disadvantages of the η-algorithm are:

1) The algorithm requires constant conduct of statistics of throughput and delay for the sources and targets in NoC. More sophisticated IP-cores (responsible for the collection of statistical data) have to be integrated in the network interfaces, to accomplish this task. This increases the chip area occupied by the network. Accordingly, the entire SoC production cost increases.
2) The algorithm deals with a large search space, when dealing with big NoCs consisting of many RecCells. This problem can be solved by using, e.g., simulated annealing. However, there is no guaranty of finding the optimal solution by the η-algorithm.

The η-algorithm is not very suitable for implementation in SoC, due to the disadvantage 1. However, it can be used in simulations. The designer can evaluate the effectiveness of other reconfiguration algorithms, comparing their results with the η-algorithm outcome.

C. The Minimal Queues Algorithm

Analysis of the various NoCs shows that the more effectively the network works, the shorter are the queues in the network buffers. Bottlenecks cause the queues in buffers on the respective network sections to rise. Subsequently, this effect generally leads to an increase of the length of the buffer queues in the entire network.

The idea of Minimal Queues Algorithm (MQA) is to react on increases of the lengths of the buffer queues in the network, and thereafter minimize these using reconfigurations. Observing the length of the buffer queues in a real SoC is much easier than keeping statistics of throughput and delay for sources and targets. Thus, the MQA is more suitable for implementation in SoC than the η-algorithm.

The trigger condition for the MQA is that the total number of packets in the network buffers exceeds some threshold specified by the developer. After that the MQA performs k reconfiguration steps. In each step, the MQA looks for switching the mode of one single RecCell that clearly shortens the lengths of the buffer queues in the entire NoC. Thereby, the MQA begins at the RecCell with the longest buffer queues. (The number k is specified by the developer. We set k equal to half of the amount of RecCells used in a network, i. e., if a network consists of four RecCells $k = 2$). The reconfiguration process requires neither to stop the operation of the NoC nor to release it entirely from packets, according to technique shown in [9].

The MQA written in pseudo-code is given below:

```
INPUT:  RecMIN, traffic;
OUTPUT: RecMIN_topology;

best_calculated_buffer_sum:= calculate(buffer_sum)
BEGIN
 IF buffer_sum>buffer_sum_threshold THEN
    IF no reconfiguration is running THEN
       FOR i:=0 TO i<k - 1 DO
        list_of_tried_cells:={};
        FOR each RecCell DO
            switching_cell:= search for
               RecCell with
               the highest  buffer_sum_in_cell;
            IF switching_cell
                 ∉ list_of_tried_cells THEN
            switch RecCell mode  (swisching_cell);
            simulate topology;
            calculate(buffer_sum);
      #if the buffer queues does not decrease
            IF NOT (calculated buffer_sum <<
                 best_calculated_buffer_sum)
            THEN
               step back to previous topology;
               add  switching_cell  to
                       list_of_tried_cells;
            END IF;
       END FOR;
       END FOR;
    END IF;
 END IF;
 RETURN actual_topology;
END;
```

Advantages and Disadvantages of MQA: As mentioned, the MQA is more suitable for real SoC than the η-algorithm, because it uses information of buffer occupation, instead of throughput and packet delay values. Furthermore, the MQA does not use an exhaustive search of all possible reconfigurations. In worst case $k * N$ reconfiguration steps have to be performed.

The main disadvantage of the MQA is that it does not provide the optimal solution. (The MQA is an empirical algorithm). Fig. 9 presents the comparison of the network performance between the η-algorithm and the MQA. Only in one of six cases, the MQA did not find the global, but the local optimum (traffic 5).

D. The Pattern Identification Algorithm

Generally, the occurrence of a bottleneck in RecCell can be identified by the occupation of its buffers. So, if such a situation arises in a particular RecCell, the overloaded channel can be diagnosed by a pattern of the buffer queues in the RecCell. Accordingly, if one of those patterns is recognised during an operation of the NoC, the RecCell has to be reconfigured.

A Pattern Identification Algorithm (PIA) can be implemented. It monitors bottleneck occurrence by recognition of buffer occupation patterns in each RecCell and performs the required reconfiguration. If more than one pattern is identified, the PIA gives priority to RecCells according to the distance of their position to the targets.

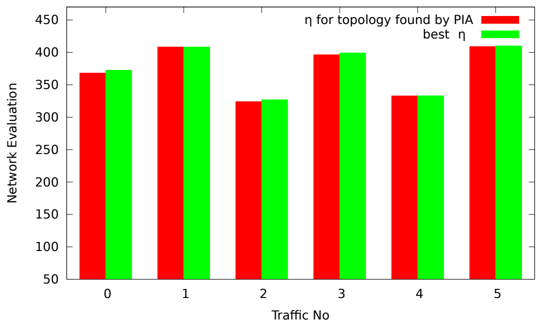

Figure 10. Comparison between PIA and η-algorithm for different traffics in 16x16 RecMIN

The PIA written in pseudo-code is given below:

```
INPUT:  RecMIN, traffic;
OUTPUT: RecMIN_topology;

BEGIN
 FOR i:=stage_number -1 DOWNTO 0 DO
   FOR each stage in RecMIN
             beginning from stage[i]  DO
     FOR each RecCell in this stage DO
      IF no reconfiguration is running THEN
        IF  is one of the patterns found  THEN
         reconfigure the cell according to the found pattern
        END IF;
      END IF;
     END FOR;
   END FOR;
 END FOR;

 RETURN actual_topology;
END;
```

Advantages and Disadvantages of PIA: An important advantage of the algorithm is that it does not search for a new topology by traversation of possible solutions. The PIA performs a reconfiguration only if it clearly improves the network efficiency. In worst case the PIA would do $2^n * k$ (where n is the index of stages in RecMIN, and k is the number of RecCells in each RecCell). But, in normal cases, the PIA is more efficient than the MQA comparing the number of reconfiguration steps.

Furthermore, a pattern search algorithm like the PIA uses buffer states as trigger information. This makes the implementation of the PIA in SoC simple. Also, the PIA performs reconfiguration steps for RecCells of the same RecMIN stage simultaneously so increasing the speed of the reconfiguration process.

The disadvantage of the PIA is that it requires implementation of additional memory registers in order to store the patterns in the NoC. Also, in case of miscarrying implementation of patterns, the RecMIN can become instable. Thus, the PIA will constantly detect one of the implemented patterns and fulfil infinite reconfiguration processes.

The PIA is the best of three algorithms proposed in this paper, for hardware realisation in SoC (in case that the patterns for the PIA are well implemented). So, for all of the six traffic flows that were used to test the performance of the three proposed algorithms, the PIA found an optimal NoC topology (Fig. 10).

V. Conclusion

In this paper, three reconfiguration algorithms were employed and evaluated, in order to benefit from the special capabilities of the Reconfigurable Multi-Interconnection Network (RecMIN) architecture. The η-algorithm, the minimal queues algorithm (MQA) and the pattern identification algorithm (PIA) allow the network to adapt itself to different traffic distributions. We evaluated the performance of the proposed reconfiguration algorithms with six chosen traffic flows and discussed the advantages and disadvantages of each algorithm. Finally, the η-algorithm is the best one for simulation. Therefore, the designer can evaluate the effectiveness of other reconfiguration algorithms, comparing their results with the η-algorithm outcome. However, the pattern identification algorithm is the most suitable reconfiguration algorithm for hardware realization in SoC.

References

[1] J. Owens, W. Dally, R. Ho, D. Jayasimha, S. Keckler, and L.-S. Peh, “Research challenges for on-chip interconnection networks,” Micro, IEEE, vol. 27, no. 5, Sept.-Oct. 2007, pp. 96 –108.

[2] D. Lüdtke, D. Tutsch, A. Walter, and G. Hommel, “Improved performance of bidirectional multistage interconnection networks by reconfiguration,” in Proceedings of 2005 Design, Analysis, and Simulation of Distributed Systems (DASD 2005); San Diego. SCS, Apr. 2005, pp. 21–27.

[3] D. Lüdtke and D. Tutsch, “Lossless static vs. dynamic reconfiguration of interconnection networks in parallel and distributed computer systems,” in Proceedings of the 2007 Summer Computer Simulation Conference (SCSC’07); San Diego. SCS, Jun. 2007, pp. 717–724.

[4] ——, “The modeling power of CINSim: Performance evaluation of interconnection networks,” Computer Networks, vol. 53, no. 8, 2009, pp. 1274–1288.

[5] M. Al Faruque, T. Ebi, and J. Henkel, “ROAdNoC: Runtime observability for an adaptive network on chip architecture,” in Computer-Aided Design, 2008. ICCAD 2008. IEEE/ACM International Conference on, Nov. 2008, pp. 543–548.

[6] ——, “Configurable links for runtime adaptive on-chip communication,” in Design, Automation Test in Europe Conference Exhibition, 2009. DATE ’09., april 2009, pp. 256 –261.

[7] A. Logvinenko and D. Tutsch, “A reconfiguration technique for area-efficient network-on-chip topologies,” in Performance Evaluation of Computer Telecommunication Systems (SPECTS), 2011 International Symposium on, June 2011, pp. 259 –264.

[8] ——, “Recsim - a simulator for reconfigurable network on chip topologies,” in Proceedings of the 26th European Simulation and Modelling Conference (ESM 2012). Essen, Germany, October 2012, pp. 144–151.

[9] A. Logvinenko, C. Gremzow, and D. Tutsch, “RecMIN: A reconfiguration architecture for network on chip,” in Reconfigurable and Communication-Centric Systems-on-Chip (ReCoSoC), 2013 8th International Workshop on, 2013, pp. 1–6.

[10] D. Tutsch, Performance Analysis of Network Architectures, 1st ed. Berlin: Springer Verlag, 2006.

[11] P. C. Wong and M. S. Yeung, “Design and analysis of a novel fast packet switch–pipeline banyan,” IEEE/ACM Transactions on Networking, vol. 3, no. 1, Feb. 1995, pp. 63–69.

[12] T.-Y. Huang and J.-L. C. Wu, “Alternate resolution strategy in multistage interconnection networks,” Parallel Computing, vol. 20, 1994, pp. 887–896.

[13] N. Boot, “Throughput and delay analysis for a single router in networks on chip,” Master’s thesis, Technische Universiteit Eindhoven, Netherlands, 2005.